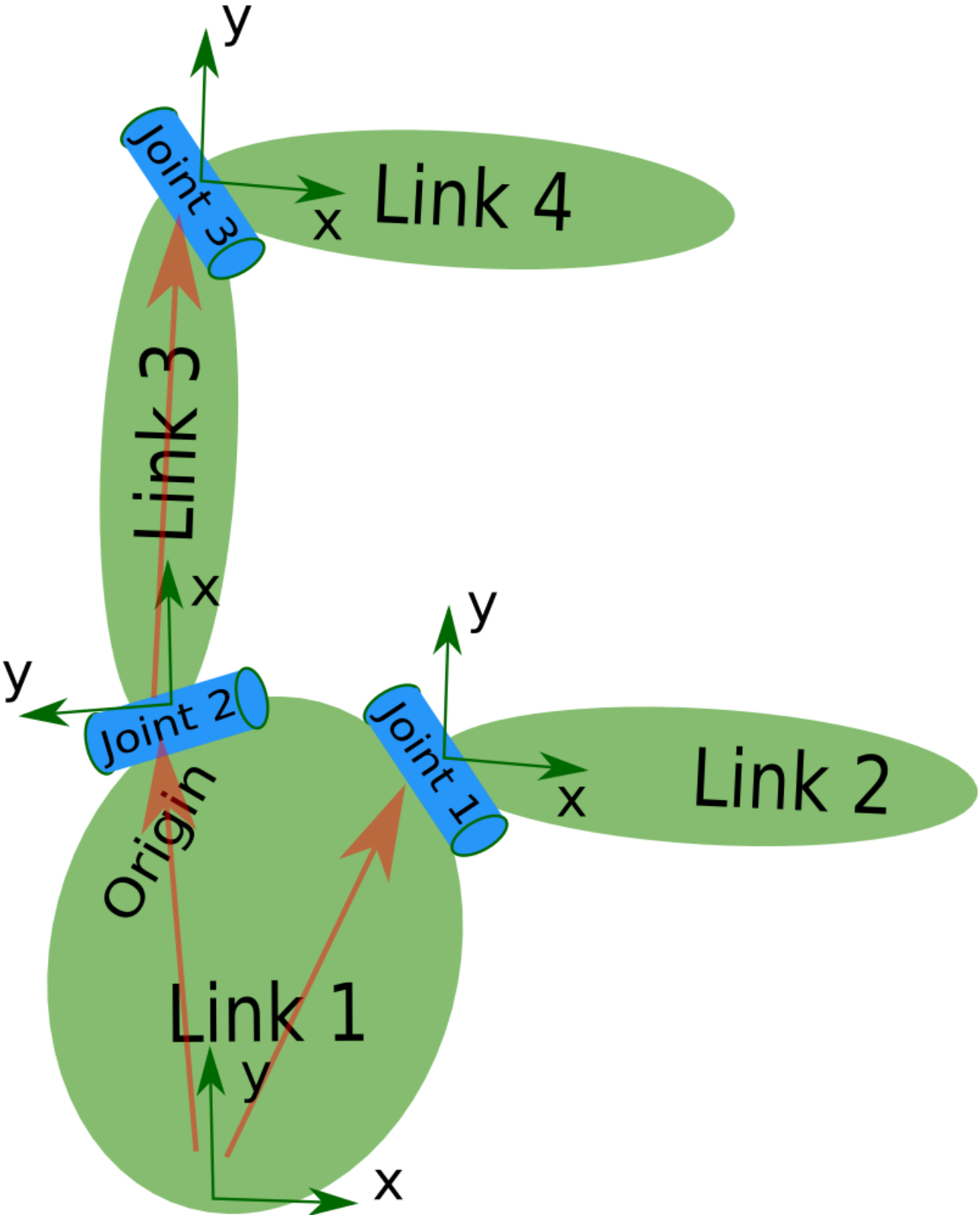
Ros建模

robot_model

Create your own urdf file

description: in this tutorial you start creating your own urdf robot description file.

- Create the tree structure
  in this tutorial we will create the URDF description of the "robot" shown in the image below.



The robot in the image is a tree structure.Let's start very simple,and create a description of that tree structure,without worrying about the dimensions etc.Fire up your favorite text editor,and create a file called my_robot.urdf:

```
1 <robot name="test_robot">
2   <link name="link1" />
3   <link name="link2" />
4   <link name="link3" />
5   <link name="link4" />
6
```

```
  <joint name="joint1" type="continuous">
    <parent link="link1"/>
    <child link="link2"/>
  </joint>

  <joint name="joint2" type="continuous">
    <parent link="link1"/>
    <child link="link3"/>
  </joint>

  <joint name="joint3" type="continuous">
    <parent link="link3"/>
    <child link="link4"/>
  </joint>
</robot>
```

So,just creating the structure is very simple.Now let's see if can get this urdf file parsed.There is a simple command line tool that will parse a urdf file for you, and tell you if the syntax is correct.
you might need to install, urdfdom as an upstream,ROS independent package:\

```
sudo apt-get install liburdfdom-tools
```

Now run the check command:

```
rosmake urdfdom_model
check_urdf my_robot.urdf
```

Add the dimensions

So now that we have the basic tree structure.let's add the appropriate dimensions.As you notice in the robot image,the reference frame of each link (in green) is located at the bottom of the link.and is identical to the reference frame of the joint.So,to add dimensions to our tree,all we habe to specify is the offset from a link to the joint(s) of its children.To accomplish this,we will add the field to each of the joints.
Let's look at the second joint.Joint2 is offset in the Y-direction from link1,a little offset in the negative X-direction from link1, and it is rotated 90 degrees around the Z-axis.So,we need to add the following element:

```
<origin xyz="-2 5 0" rpy="0 0 1.57"/>
```

If you repeat this for all the elements our URDF will look like this:

```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
```

```
  <link name="link3" />
  <link name="link4" />


  <joint name="joint1" type="continuous">
    <parent link="link1"/>
    <child link="link2"/>
    <origin xyz="5 3 0" rpy="0 0 0" />
  </joint>

  <joint name="joint2" type="continuous">
    <parent link="link1"/>
    <child link="link3"/>
    <origin xyz="-2 5 0" rpy="0 0 1.57" />
  </joint>

  <joint name="joint3" type="continuous">
    <parent link="link3"/>
    <child link="link4"/>
    <origin xyz="5 0 0" rpy="0 0 -1.57" />
  </joint>
</robot>
```

Completing the Kinematics

What we didn't specify yet is around which axis the points rotate.Once we add that,we actually have a full kinematic model of this robot!All we need to do is add the element to each joint.The axis specifies the rotational axis in the local frame.
So, If you look at joint2, you see it rotates around the positive Y-axis.So, simple add the following xml to the joint element:

```
<axis xyz="0 1 0" />
```

Similarly,joint1 is rotating around the following axis:

```
<axis xyz="-0.707 0.707 0"/>
```

Note that it is a good idea to normalize the axis. If we add this to all the joints of the robot, our URDF looks like this:

```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

```

```
  <joint name="joint1" type="continuous">
    <parent link="link1"/>
    <child link="link2"/>
    <origin xyz="5 3 0" rpy="0 0 0" />
    <axis xyz="-0.9 0.15 0" />
  </joint>

  <joint name="joint2" type="continuous">
    <parent link="link1"/>
    <child link="link3"/>
    <origin xyz="-2 5 0" rpy="0 0 1.57" />
    <axis xyz="-0.707 0.707 0" />
  </joint>

  <joint name="joint3" type="continuous">
    <parent link="link3"/>
    <child link="link4"/>
    <origin xyz="5 0 0" rpy="0 0 -1.57" />
    <axis xyz="0.707 -0.707 0" />
  </joint>
</robot>
```

Update your file my_robot.urdf and run it through the parser.

```
check_urdf my_robot.urdf
```

That's it.you created your first URDF robot description!Now you can try to visualize the URDF using graphiz:

```
urdf_to_graphiz my_robot.urdf
```

and open the generated file with your favorite pdf viewer:

```
evince test_robot.pdf
```

Parse a urdf file

Description: This tutorial teachs you how to use the urdf parser

Reading a URDF file

This tutorial starts off where the previous one ended.You should still habe your my_robot.urdf file with a description of the robot shown before below.
Let's first create a package with a dependency on the urdf parser in our snadbox:

```

```

```
cd ~/catkin_ws/src
catkin_create_pkg testbot_description urdf
cd testbot_description
```

Now create a /urdf folder to store the urdf file we just created:

```
mkdir urdf
cd urdf
```

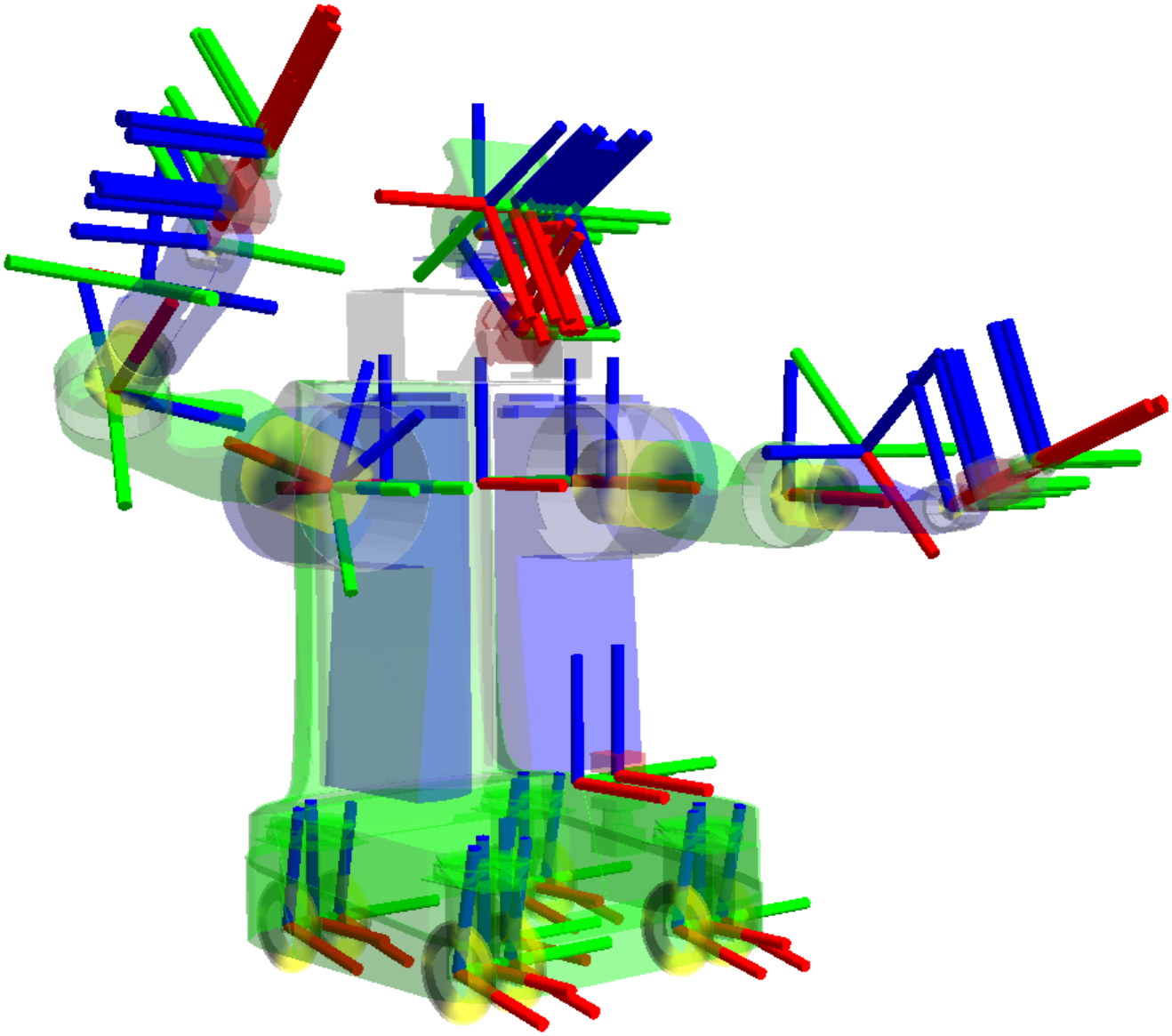
This follows the convention of always storing your robot's URDF file in a ROS package named MYROBOT_description and within a subfolder named /urdf.Other standard subfolders of your robot's description package include /meshes, /media and /cad,like so:\

```
/MYROBOT_description
  package.xml
  CMakeLists.txt
  /urdf
  /meshes
  /materials
  /cad
```

Using the robot state publisher on your own robot

Description: This tutorial explains how you can publish the state of your robot to tf, using the robot state publisher.

When you are working with a robot that has many relevant frames.it becomes quite a task to publish them all to tf.The robot state publisher is a tool that will do this job for you.



The robot state publisher helps you to broadcast the state of your robot to the tf transform library.The robot state publisher internally has a kinematic运动学 model of the robot; so given the joint positions of the robot,the robot state publisher can compute and broadcast the 3D pose of each link in the robot. You can use the robot state publisher as a standalone ROS node or as a library:

1.Running as a ROS node

1.1 robot_state_pubisher

The easiest way to run the robot state publisher is as a node.For normal users,this is the recommanded usage.You need two things to run the robot state publisher:

> - a urdf xml robot description loaded on the Parameter Server.
> - A source that publishes the joint positions as a sensor_msgs/JointState

1.1.1 Subscribed topics

joint_states(Sensor_msgs/JointState)

> joint position information

1.1.2 Parameters

robot_description(urdf map)
tf_prefix(string)

> Set the tf prefix for namespace-aware publishing of transform publish_frequency(double) Publish frequency of state publisher,default:50Hz

1.2 Example launch file

```
<launch>
  <param name = "my_robot_description" textfile = "$(find
mypackage)/urdf/robotmodel.xml"/>
  <node pkg = "robot_state_publisher" type="robot_state_publisher" name =
"rob_st_pub">
  <remap from="robot_state_publisher" to="my_robot_description"/>
  <remap from="joint_states" to="different_joint_states"/>
  </node>
</launch>
```

2.Runing as a library

Advanced users can also run the robot state publisher as a library,from within their own c++ code.After you include the header: `#include <robot_state_publisher/robot_state_publisher.h>`
all you need is the constructor which takes in a KDL tree
`RobotStatePublisher(const KDL::Tree& tree);`
and now, everytime you want to publish the state of your robot, you call the publishTransforms functions:

```
//Publish moving joints
void publishTransfroms(const std::map<std::string, double>&
joint_positions,
  const ros::Time& time);

//publish fixed joints
void publishFixedTransforms();
```

The first argument is a map with joint names and joint positions, and the second argument is the time at which the joint positions were recorded. It is okay if the map does not contain all the joint names. It is also okay if the map contains some joints names that are not part of the kinematic model. But note if you don't tell the joint state publisher about some of the joints in your kinematic model, then your tf tree will not be complete.

Using urdf with robot_state_publisher

Create the URDF File

Publishing the State

> cd %TOP_DIR_YOUR_CATKIN_WS%/src catkin_create_pkg r2d2 roscpp rospy tf sensor_msgs std_msgs Then fire your favourite editor and paste the following code into the src/state_publisher.cpp file: